

Aprendizado de Máquina - Trabalho

Universidade Federal de Santa Catarina

Departamento de Automação e Sistemas

Prof. Eric Aislan Antonelo

Observações importantes:

1. Avaliação: seu código será avaliado por corretude técnica, e por alcançar resultados similares aos esperados para os itens pedidos.
2. Desonestidade acadêmica: seu código será checado através softwares capazes de detectar redundância lógica contra outras submissões e arquivos obtidos da internet. Caso seja confirmada a fraude/**plágio**, medidas cabíveis serão tomadas, incluindo **nota zero** no trabalho.
3. No relatório, cite as referências/bibliografia usadas.

Opção 1 - Treinamento de redes neurais multi-camadas (Recomendada)

Neste trabalho, você deverá implementar o método da retropropagação (*backpropagation*) para o cálculo do gradiente da função de custo com relação aos pesos de uma rede neural multi-camadas (número de camadas ocultas $n_o \geq 2$).

Se implementado em Python, use a biblioteca *numpy* (`np.dot`, `np.array`, etc.).

1. Implementação:

- a) Crie as estruturas de dados para guardar os pesos que definem uma arquitetura de rede neural multi-camadas. Inicialize a rede neural aleatoriamente.
- b) Implemente o algoritmo da retropropagação para o cálculo do gradiente, a derivada parcial da função de custo com relação aos pesos da rede.
- c) Valide o algoritmo do cálculo do gradiente, realizando uma aproximação numérica do mesmo. Verifique se os cálculos batem um com o outro.
- d) Dado o gradiente já calculado, implemente o método do descenso do gradiente para o treinamento da rede neural, ou seja, o processo de ajuste dos pesos.

2. Aplicação:

- a) Use o código implementado para treinar uma rede neural para realizar a classificação de um padrão de duas dimensões de entrada. Os dados para treinamento estão disponíveis no arquivo

`classification2.txt`.

Para plotar a fronteira de decisão da rede treinada, poderá usar o código disponível no link

https://colab.research.google.com/drive/1XTtZGgpAefbiWejTrEjsnWzS_XXYdzff?usp=sharing.

- b) Relate resultados variando pelo menos duas vezes cada um dos hiperparâmetros: o número de camadas; o número de neurônios por camada; taxa de aprendizagem. Use métricas como taxa de classificação (porcentagem de predições corretas) no conjunto de validação (exemplos não usados no treinamento).
- c) (opcional) Treine uma rede neural para classificar dígitos a partir de imagens como entrada para a rede. Use o arquivo `classification3.mat`.

3. Entregas:

No **relatório** a ser entregue, descreva os experimentos e os resultados obtidos.

Na apresentação, você deverá explicar o código implementado de uma forma geral, as dificuldades encontradas, e em especial:

- a) a parte do código referente ao cálculo do gradiente;
- b) a parte do código referente ao *gradient descent*;
- c) gráfico da função de custo e taxa de classificação no conjunto de treinamento e no conjunto de validação ao longo das épocas (de treinamento);
- d) taxa de classificação no conjunto de validação/teste, após finalizado o treinamento;
- e) o gráfico da fronteira de decisão;
- f) outros resultados, por ex., sensibilidade aos hiperparâmetros.

Entregue o código da implementação e o PDF do relatório pelo Moodle (zipado com ZIP ou tar.gz; RAR não é suportado).

Opção 2 - Clonagem comportamental (Alternativa)

Neste trabalho, você usará frameworks de *deep learning* para treinar redes neurais profundas a fim de controlar veículos ou robôs através da clonagem comportamental.

Aprendizado por imitação é uma abordagem pela qual um modelo caixa-preta (rede neural) é treinado para imitar um especialista usando um conjunto fixo de amostras de pares observação-ação (ou trajetórias) obtidas daquele especialista. A **clonagem comportamental (CC)** é um tipo de aprendizagem por imitação baseada em um processo de treinamento supervisionado de um modelo (rede neural) usando um grande conjunto de dados rotulados. A CC tem sido utilizada para a obtenção de políticas de condução autônoma para veículos, onde as amostras de treinamento são geradas por motoristas humanos: a entrada da rede neural é a imagem da câmera do carro, enquanto a saída desejada corresponde ao atuador (ação do motorista: aceleração, direção, freio).

1. Simulação e Coleta de Dados:

- (a) Escolha um simulador de veículos ou robôs para coletar dados de treinamento para realizar a clonagem comportamental. Defina o problema de aprendizagem, incluindo o comportamento desejado, os sensores e atuadores do agente.

Vocês podem escolher o CarRacing-v2, que é um problema de controle contínuo a partir de pixels mais fácil:

https://gymnasium.farama.org/environments/box2d/car_racing/#car-racing

A versão 0 CarRacing-v0 estava disponível pelo OpenAI Gym, cujo código fonte está disponível em:

https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py)

Outros simuladores:

- i. <https://github.com/carla-simulator/carla>
 - ii. <http://gazebo.org/>
 - iii. <https://cyberbotics.com/>
- (b) Colete os dados de treinamento, em forma de um conjunto de trajetórias do comportamento desejado. Para isso, deverá controlar o veículo/robô manualmente, fornecendo sua própria ação de controle humana (especialista), que servirá para demonstrar o comportamento desejado para rede neural. Se possível, gere trajetórias dessa maneira (joystick/teclado). Senão, crie um controlador (*autopilot*) em modo *God* (que tem acesso global ao simulador, por ex., sensores GPS, etc.) que fará esse papel de gerar trajetórias de exemplo. Acrescente ruído na geração dos dados com o simulador, que pode ajudar na generalização da rede neural durante o treinamento.

2. Treinamento e Avaliação:

- (a) Uma vez escolhido o simulador e agente, defina a arquitetura de rede neural, usando *pyTorch* ou *tensorflow/keras*. Use redes convolucionais para processar entradas do tipo imagem.
- (b) Treine a rede neural com os dados de treinamento. Ao longo do treinamento, avalie seu desempenho plotando a função de custo ao longo do treinamento para os dados de treinamento e para dados de validação. Em especial, **avaliar a rede controlando o agente no simulador** (pode usar o *score* obtido do simulador).
- (c) Analise os resultados obtidos ao variar os hiperparâmetros da rede neural: número de camadas de convolução/pooling; tamanho do kernel; tamanho do conjunto de treinamento; com ou sem *dropout*, etc. O desempenho do agente no simulador (*score*) piora/melhora? Use métricas.
- (d) Defina um segundo comportamento diferenciado, treine e avalie a rede neural para o mesmo, conforme foi feito para o primeiro comportamento.
- (e) (*opcional*) Plote as saídas da rede neural para o conjunto de validação (dados estáticos; sem feedback do simulador) e para quando a rede treinada controla o agente no simulador (com feedback). Pode usar plot normal e histograma, por exemplo. Os dois casos diferem?

3. Entregas:

No **relatório** a ser entregue, descreva os experimentos e os resultados obtidos. Explique sua abordagem, arquitetura da rede, e apresente os gráficos da função de custo ao longo do treinamento. Use uma ou mais métricas para avaliar os agentes (rede neural) durante a navegação ou reprodução de comportamento (distância média percorrida sem colisões ou sem sair da estrada, tempo para conclusão da trajetória, etc.).

Na apresentação, você deverá explicar o código implementado de uma forma geral, as dificuldades encontradas, e em especial:

- a) a parte do código referente a coleta de dados, sensores e atuadores;
- b) a parte do código referente a rede neural;
- c) e mostrando a rede neural controlando o agente na simulação.

Entregue o código da implementação e o PDF do relatório pelo Moodle (zipado com ZIP ou tar.gz; RAR não é suportado).

Mais detalhes sobre CARLA, rede neural e aprendizado por imitação (ver seções 3.2 Imitation learning e S.2.2 Imitation Learning): <http://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf>

Links para códigos de referência em *pyTorch*:

1. Regressão e CNNs em *pyTorch*:

https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html

<https://medium.com/@benjamin.phillips22/simple-regression-with-neural-networks-i>

<https://www.guru99.com/pytorch-tutorial.html>

2. Clonagem comportamental para controle de um ponto em um círculo: https://colab.research.google.com/drive/1IWRgLeTug-7NphtB54iDz8aJEi_OpWbQ?usp=sharing

Problema explicado no artigo *InfoGAIL* (Figura 1): <https://arxiv.org/abs/1703.08840>